Parallel Speed-Up of Monte Carlo Methods for Global Optimization

by

R. Shonkwiler
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332
e-mail: shenk@math.gatech.edu

and

Erik Van Vleck
Department of Mathematics
Simon Fraser University
Burnaby, BC V5A 1S6 Canada
e-mail: vvleck@cs.sfu.ca

This paper is dedicated to the memory of Stanislaw Ulam

Running head: Parallel Monte Carlo Methods

Mail proofs to:
R. Shonkwiler
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We introduce the notion of expected hitting time to a goal as a measure of the convergence rate of a Monte Carlo optimization method. The techniques developed apply to Simulated Annealing, Genetic Algorithms and other stochastic search schemes. The expected hitting time can itself be calculated from the more fundamental complementary hitting time distribution (CHTD) which completely characterizes a Monte Carlo method. The CHTD is asymptotically a geometric series, $\frac{1/s}{1-\lambda}$, characterized by two parameters, s, λ, related to the search process in a simple way. The main utility of the CHTD is in comparing Monte Carlo algorithms. In particular we show that independent, identical Monte Carlo algorithms run in parallel, IIP parallelism, exhibit superlinear speedup. We give conditions under which this occurs and note that equally likely search is linearly spedup. Further we observe that a serial Monte Carlo search can have infinite expected hitting time, but the same algorithm when parallelized can have finite expected hitting time. One consequence of the observed superlinear speedup is an improved uni-processor algorithm by the technique of in-code parallelism.

List of Symbols

σ	sigma	ρ	rho
α	alpha	ϵ	epsilon
δ	delta	Δ	upper case delta
θ	theta	Θ	upper case theta
ϕ	phi	$\emptyset$	upper case phi (empty set)
ω	omega	λ	lambda
χ	chi	τ	tau
$\mathcal{P}$	script P	$\mathcal{W}$	script W
$\mathcal{X}$	script X	ℓ	script ell
$\mathbf{1}$	boldface one		

1 Introduction

In this article we will be concerned with the global optimization of a scalar-valued function $v = f(x)$, $x \in D$, defined on some set D. The domain D need not have a natural topological structure, but any search scheme will lead to a neighborhood topology, hence an existing topology, if present, can be utilized. As we will be interested in computational solutions, we impose the condition that D be a finite set although we make no restriction on its cardinality. Thus D might be the set of all n-tuples of computer floating point numbers. We also make no assumption about the smoothness or even the continuity of f since our methods use only the values v. In fact, even if advantage is taken of descent techniques available for identifying local optimizers for smooth objective functions, these methods are only successful at locating a global optimizer if the starting point of the search is judiciously chosen. For complicated functions or those with many variables, this becomes the key issue and it leads to an undirected search problem quite similar to that for non-smooth objectives.

Given then the undirected nature of a global search, it is not surprising that random or partially randomized searches have enjoyed a measure of success. Over the last few years new and powerful Monte Carlo methods for optimization have been developed and investigated. They are proving to be particularly well-suited for objective functions arising from combinatorial problems and objective functions dependent upon a large number of parameters, especially when the dependence gives rise to numerous local optima. The well-known NP-complete problems and various other problems arising in computer science are of the former type while problems of the latter type pervade engineering and science.

A particularly rich source of applications arise in an attempt to solve "inverse" problems such as occur in oil reservoir location, and in fractal image processing, (DARPA, 1987). Here the "forward" equations are well-known and easy to solve, but these equations can not be tractably inverted. Thus in oil reservoir location, if the subterranean structure were known, then a detonation return echo signature can be computed. But

given a set of return echoes, the only way to predict oil reservoir locations is to guess at them and confirm it by comparing the predicted and observed echoes. Generally, for such inverse problems, error between the forward resultant and the goal is used as an objective function.

By Monte Carlo optimization we mean any method that utilizes random number generation in some aspect of its search for a global optimizer. Prominent among such methods are two inspired by natural phenomena, simulated annealing and genetic algorithms. The former is an abstraction of a thermodynamic process and falls within the purview of statistical mechanics. Random number generation is used to construct new abstract thermodynamic states via Boltzmann dynamics, (van Laarhoven and Aarts, 1987), (Kirkpatrick, Gelatt, and Vecchi, 1983). The latter is a paradigm of natural evolution. Here random number generation is used to simulate matings, mutations, and survival in such a way that reproduction is enhanced for the "fittest," (Holland, 1975), (Goldberg, 1989). The motivating and guiding force behind these numerical methods is the success of their respective natural phenomena at reaching optimal states. Thus annealed physical systems achieve minimal states of internal energy and evolution produces organisms capable of species survival.

Our notion of Monte Carlo optimization includes these, but also admits search techniques having no phenomenological basis.

Virtually all optimization processes are iterative proceeding in steps indexed by "time" $t = 0, 1, 2, \ldots$. On the t^{th} step one or more domain points x are selected to constitute the current *state* X_t of the process

$$X_t = \{x_1^t, \ldots, x_{n(t)}^t\} \subset D,$$

and any required function evaluations, $v_i^t = f(x_i^t)$, are performed. The sequence of random variables $X_0, X_1, X_2, \ldots$ is a stochastic process on the class of all finite subsets of D. Let

D_{op} denote the subset of the domain consisting of the optimal points. If the problem is one of global minimization, then

$$D_{op} = \{x_* \in D : f(x_*) \leq f(x), x \in D\}.$$

Most optimization algorithms base the next trial set of optimizers X_{t+1} on the last set X_t or maybe the last two sets. It is practically required that an algorithm save only a fixed number of previous trials, as evidently, saving all the information $X_0, X_1, \ldots$, will eventually exceed the finite storage capacity of the machine. In the event that X_{t+1} only depends on X_t, then the stochastic process is a Markov chain and its analysis is greatly simplified. If the next iteration depends on some fixed finite number of previous iterations, say j, this too can be treated as a Markov chain by regarding the Markov chain states as the j-tuples, $(X_t, X_{t-1}, \ldots, X_{t-j+1})$. In the following we assume that X_{t+1} depends only on X_t. Both simulated annealing and genetic algorithms are of this type.

Also we will assume that the *current best* random variable, B_t, is monitored throughout the optimization process. This is defined as the single domain point b where

$$B_t = b \in \bigcup_0^t X_k \quad \text{such that} \quad f(b) \leq f(x),\ x \in \bigcup_0^t X_k. \tag{1.1}$$

In case of ties, i.e. $f(b) = f(x)$ for some x, b is the point encountered first in the chain.

A top level algorithm then takes the form of fig. 1. In general, the next state X_{t+1} of the chain can depend on time t as well as the set of domain points X_t. In the event that the transition does not depend on t, the chain is said to be *stationary*, otherwise it is *non-stationary*.

Hitting Time Problem

Any method which attempts to locate global optima must address three important issues: (1) will the globally best value be found by the method, (2) how can the globally best value be identified as such, and (3) how long will it take to reach it? Of course the

Top Level Algorithm

<calculate the initial set of trial solutions X_0 and their function values>

$t \leftarrow 1$

repeat

  <using X_t, and possibly t, stochastically generate a new set

    of trial solutions, X_{t+1}, and their function values>

  <update B_t >

  increment *time*

until <the stop criteria is met>

< report B_t >

fig.1

techniques for dealing with these problems depend greatly on what may be known about the objective function itself. In the extreme event that it has no regularity whatsoever, then the globally maximum and minimum values will be found and known for certain only by *exhaustion*, that is by examining every domain point. Consequently theoretical results about convergence often require that the number of iterations grow without bound to assure a solution. For simulated annealing we have the following due to (Hajek, 1988), (Geman and Geman, 1984), (Gidas, 1985)

$$\lim_{t \to \infty} P_r(X_t \cap D_{op} \neq \emptyset) = 1, \tag{1.2}$$

provided that certain conditions on the search are met. Thus the probability is high that the process has found, and is in, an optimal state only after "long" run times.

In view of this difficulty, most implementations either (a) run for a pre-specified length of time and accept the best value observed over the run, or (b) assign a pre-determined achievable value v_g as a goal, selected from previous familiarity with the problem, and stop

upon reaching it. An important class of applications of the latter type are those in which the objective function measures error and it is known that zero error is possible; rather it is the optimizer that is of interest.

It should be mentioned at this point that there is a body of theory specific to the stopping time problem, see (Chow, Robbins, and Siegmund, 1971), (Dorea, 1990), (Zuckerman, 1986). And so we will not deal with the stopping issue here. Instead the present article addresses only the third question, how long can we expect it to take before reaching the goal. We assume there is some definite non-empty subset of domain points constituting the specified goal, say all points $u \in D$ such that $f(u) \leq v_g$. The set of goal domain points identifies with a set of Markov chain states which become the Markov chain goal, G, namely any Markov chain state that contains a goal domain point u. Of course, by taking $v_g = f(x_*)$, then $f(u) \leq v_g$ if and only if $u \in D_{op}$ and so this scheme includes the original global minimization problem.

The question as to how long it will take to reach a goal state translates into an expected hitting time question for Markov chains about which much is known, (Chung, 1967), (Kemeny and Snell, 1960). Specifically, by the *hitting time* in G we mean the random variable θ equal to the first time t so that

$$X_t \in G.$$

In terms of hitting time, a process will eventually find an optimal state if

$$\Pr(\theta < \infty) = 1.$$

By definition, the expected hitting time is given by

$$E(\theta) = \sum_{t=1}^{\infty} t \Pr(\theta = t) \tag{1.3}$$

where $\Pr(\theta = t), t = 1, 2, \ldots$, is the probability density function for θ. Since

$$\Pr(\theta = t) = \Pr(\theta \geq t) - \Pr(\theta \geq t + 1),$$

direct substitution in (1.3) yields the following alternative equations

$$E(\theta) = \sum_{t=1}^{\infty} \Pr(\theta \geq t) = \sum_{t=1}^{\infty} (1 - F(t)) \tag{1.4}$$

where $F(t) = \Pr(\theta < t)$ is the cumulative distribution function for θ. The all important *complementary hitting time* distribution

$$\{\Pr(\theta \geq t)\}_{t=1}^{\infty}$$

characterizes the optimization process and occupies a central place in this work. Evidently

$$0 \leq \Pr(\theta \geq t) \leq 1, \qquad t = 1, 2, \ldots,$$

and is a monotone decreasing sequence

$$\Pr(\theta \geq t) \geq \Pr(\theta \geq t + 1), \qquad t = 1, 2, \ldots,$$

see fig. 3 and 5.

An important use of the complementary hitting time distribution is in the comparison of optimization processes. From their distributions, the expected hitting time for two different Monte Carlo methods for a given problem or class of problems can be calculated to decide the superior. We do exactly that to analyze the parallelization of a Monte Carlo search by a method we call IIP, Independent Identical Processes.

IIP Parallel Monte Carlo

Our simple parallelization method is this, run a separate copy of the single processor algorithm, just as it is, on each of the individual multiple processors. As discussed in more detail below, there should be no communication between processors except for some mechanism to shut them all down when the first one finds the solution.

There is no problem in implementing the prescribed parallelization on various parallel platforms. Monte Carlo algorithms are typically quite simple and do not use large amounts

of memory. Consequently, as many processes can run simultaneously on a shared memory machine as there are CPU's. All the processes can start up in milli-seconds. The same goes on a massively parallel architecture such as the Connection Machine. It takes only milli-seconds to start up the parallel processes, even for 65,536 of them. Less time is required to flag a stop when some process succeeds in finding the goal. To insure that the independent processes are not performing the exact same random walk, it is only necessary to seed their individual random number generators differently. The parallel processing can also be accomplished on a LAN consisting of a single server or multiple servers. The feasible number of parallel processes for a LAN is up to a few hundred.

Superlinear Speedup

We show in section 2 that for a (single process) stationary Monte Carlo search, the expected number of iterations, E_1, required to find the goal is governed by two scalar parameters $s > 0$ and λ, $0 < \lambda < 1$, which depend on the details of the search. E_1 is given approximately by

$$E_1 = \frac{1}{s}\frac{1}{1-\lambda}.$$

But the expected number of parallel iterations required by m processors, E_m, due to the independence, is given approximately by

$$E_m = (\frac{1}{s})^m \frac{1}{1-\lambda^m}.$$

Therefore the parallel speedup, S_m, is

$$S_m = \frac{E_1}{E_m} = s^{m-1}\frac{1-\lambda^m}{1-\lambda},$$

see the Main Theorem, section 3. For λ near 1, and this is always the case, the latter expression works out to be approximately ms^{m-1}. The parameter s can range from 0 to infinity, but, as a rule, s is always bigger than 1; we say the process is *accelerated.* Loosely speaking, $s > 1$ in problems where the Monte Carlo search is not drawn towards the

optimizer, that is in *hard problems*. Therefore the speedup is, initially at least, exponential in the number of processes and it is quite possible for the parallel search to succeed more than m times faster than its serial counterpart. We term this *superlinear speedup*, see (Bertsekas and Tsitsiklis, 1989, p.15).

Therefore the implementation of a Monte Carlo search on parallel computers with a large number of processors, such as the 64K processor Connection Machine, can experience many thousand fold speedup over the single process algorithm. This is even the case if the individual processes must proceed in lock step or inter-process communication is restricted. Each processor is executing the same code, namely the serial algorithm (of course each process must be seeded differently).

In some cases a Monte Carlo algorithm will have an infinite expected hitting time. We show this can occur for non-stationary algorithms as is simulated annealing. This occurs because the sum of the complementary hitting time probabilities diverges. But we show that in this case it is possible for the same algorithm, when multi-processed, to have *finite* expected hitting time.

Finally, we will show that when complete information about the past iterations is used, one can still expect a speedup of approximately $m/2$ for m-fold parallelization.

This paper is organized as follows. In Section 2 we derive formulas for the expected hitting time and the complementary hitting time distribution for stationary and non-stationary finite Markov chains. Also we apply these results to some specific problems. In Section 3 we derive the hitting time formulas for multi-processed implementations of these Monte Carlo methods. In Section 4 we illustrate the results first on two example problems of sufficiently small size that s and λ can be exactly calculated. Their Monte Carlo implementation shows excellent agreement with predicted hitting time and parallel speedup. We also repeat here the speedup results for a Monte Carlo solution of the substantial 1-dimensional inverse fractal problem. We empirically obtain a speedup of 709 by the use of 24 processors. Finally we show by example that in the non-stationary

case, m-fold multi-processing can convert an infinite expectation process into a *finite* one (thereby achieving infinite speedup).

2 Hitting Time Calculations

2.1 *Stationary Markov Chains*

To fix ideas, let $\mathcal{W} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$ denote the collection of all subsets $\mathcal{P} \subset D$ of fixed cardinality $n \geq 1$. These are the populations or Markov chain states of the search. Populations with duplicate domain points are allowed. As noted in the introduction, we are assuming there are finitely many such states, N, although this is typically very large.

The transition probabilities at time t are

$$p_t^{ij} = \Pr(X_{t+1} = \mathcal{P}_j \mid X_t = \mathcal{P}_i)$$

and their matrix, $P(t)$, is the *transition probability matrix*. Thus the i^{th} row of $P(t)$ gives the distribution of probabilities for transitioning from $\mathcal{P}_i$ to the other states (or back to itself). Since for any given state, these transitions exhaust the possibilities, all the rows of $P(t)$ sum to 1. In general the transition probabilities vary with time t but in this subsection we assume they are fixed, $P(t) = P$, $t = 1, 2, \ldots$. We also assume, without loss of generality, that the set G of goal states, say g in number, are the first states of the enumeration $\mathcal{P}_1, \ldots, \mathcal{P}_g$. The division of states into goal and non-goal induces a corresponding division of the transition probability matrix into submatrices

$$P = \begin{pmatrix} J & H \\ B & \hat{P} \end{pmatrix}$$

where J is $g \times g$, H is $g \times N - g$, B is $N - g \times g$, and $\hat{P}$ is $N - g \times N - g$. In this B is the submatrix of one step "bridge" transitions from non-goal to goal states. Large probabilities here are desirable for quick solution. It goes without saying that B will not be the zero matrix in any reasonable search process and we will assume it is not. The submatrix $\hat{P}$ is of central importance for us because it describes the search process during

the time prior to finding a solution. We refer to it as the *deleted transition probability matrix* since it is obtained from P by deleting the rows and columns corresponding to goal states. By contrast the matrices J and H have a much diminished role due to the assumption that a goal state can be recognized as such. As a result, the search process stops upon encountering such a state and therefore the submatrices J and H may be taken to be the identity matrix and the zero matrix respectively. It should be emphasized that the transition probabilities are never explicitly determined, nevertheless they arise by consequence of the specifics of the search process.

As the search proceeds, its progress is described by the probability distribution vector, $\alpha_t = (\alpha_t^1, \ldots, \alpha_t^N)^T$ (superscript T denotes transpose), where

$$\alpha_t^i = \Pr(X_t = \mathcal{P}_i).$$

Taking α_0 as the starting distribution, the distribution after the k^{th} iteration is given by the k fold matrix product

$$\alpha_k^T = \alpha_0^T P^k.$$

As with P, we also partition α_t into its first g components, α_t^G, and its last $N-g$ components, $\hat{\alpha}_t$. Then a matrix product $\alpha_t^T P$ becomes

$$\alpha_t^T P = ((\alpha_t^G)^T + \hat{\alpha}_t^T B \mid \hat{\alpha}_t^T \hat{P}).$$

In particular

$$\hat{\alpha}_{t+1}^T = \hat{\alpha}_t^T \hat{P} = \ldots = \hat{\alpha}_0^T \hat{P}^{t+1}, \qquad t = 0, 1, 2, \ldots .$$

The partitioned calculation also shows that the new transitions into the goal states on the $(t+1)^{st}$ iteration are given by $\hat{\alpha}_t^T B$. Hence

$$\Pr(\theta = t+1) = \hat{\alpha}_t^T B \mathbf{1} \ ,$$

where $\mathbf{1}$ is the $(N-g)$ vector of 1's and has the effect of adding up the components of its companion.

Now the expected hitting time may be calculated using (1.4) as follows, counting the initialization step, that is choosing X_0, it is clear that $\Pr(\theta \geq 1) = 1$. The distribution of the initialization step is given by $\alpha_0 = (\alpha_0^G \mid \hat{\alpha}_0)^T$. Further, θ will be exactly 1 only if X_0 is a goal state. This occurs with probability equal to the sum of the terms of α_0^G or equivalently with probability $1 - \hat{\alpha}_0^T \mathbf{1}$, where again $\mathbf{1}$ is the $N - g$ vector of 1's. Therefore $\Pr(\theta \geq 2) = \hat{\alpha}_0^T \mathbf{1}$.

Next observe that the hitting time is 3 or more if and only if the process is still in the non-goal states after the second iteration. But this occurs with probability $\hat{\alpha}_1^T \mathbf{1}$ which therefore is $\Pr(\theta \geq 3)$. In general it follows in the same way that

$$\Pr(\theta \geq t) = \hat{\alpha}_{t-2}^T \mathbf{1} = \cdots = \hat{\alpha}_0^T \hat{P}^{t-2} \mathbf{1}, \quad t = 3, 4, \ldots . \tag{2.1}$$

Altogether then

$$\begin{aligned} E(\theta) &= 1 + \hat{\alpha}_0^T \mathbf{1} \ + \hat{\alpha}_1^T \mathbf{1} \ + \hat{\alpha}_2^T \mathbf{1} \ + \ldots \\ &= 1 + \hat{\alpha}_0^T \mathbf{1} \ + \sum_{t=3}^{\infty} \hat{\alpha}_0^T \hat{P}^{t-2} \mathbf{1} \ . \end{aligned} \tag{2.2}$$

Under mild conditions on the search process, the deleted transition probability matrix $\hat{P}$ will be *primitive*, that is some power of $\hat{P}$ will have all positive entries. This will hold for instance if it is always possible to transition, in some finite number of steps, from any one state to any other and if there is at least one state in which the process can stay for one, or more, iterations. (These conditions imply the process is *irreducible* and *aperiodic*.)

By the Perron-Frobenius theorem for primitive matrices, $\hat{P}$ has a positive eigenvalue λ equal to its spectral radius, $\lambda = \rho(\hat{P})$. We will refer to λ as the *principal* eigenvalue. It lies strictly between the largest and smallest of the row sums of $\hat{P}$ unless these are equal in which case λ is their common value. Now the i^{th} row sum, σ_i, of $\hat{P}$ will be the difference between 1 and the sum of the one-step transitions (in B) to a goal state from the i^{th} non-goal state, that is

$$\sigma_i = \sum_{j=1}^{N-g} \hat{p}^{ij} = \sum_{j=g+1}^{N} p^{g+i,j} = 1 - (p^{g+i,1} + \cdots p^{g+i,g}) \leq 1.$$

Since the one-step matrix B is not zero, at least one such row sum is strictly less than 1. Hence

$$\lambda < 1.$$

Let χ be a positive right eigenvector and ω a positive left eigenvector of $\hat{P}$ corresponding to λ (guaranteed by the Perron-Frobenius theorem).

$$\hat{P}\chi = \lambda\chi, \qquad \omega^T\hat{P} = \lambda\omega^T.$$

We may normalize ω and χ so that the former is a probability vector, $\sum \omega^i = 1$, and the latter so that $\omega^T\chi = 1$; then ω and χ are uniquely determined. The next result may be found in (Seneta, 1981).

Theorem. *If $\lambda > |\lambda_2|$, where the latter is the magnitude of the next largest eigenvalue after λ, then there exists an integer $1 \le d < N - g$, and a fixed polynomial $h(\cdot)$ of degree $d-1$ such that*

$$\left| \frac{1}{\lambda^k}(\hat{\alpha}_0^T \hat{P}^k \mathbf{1}\ ) - \hat{\alpha}_0^T \chi \right| < h(k) \left| \frac{\lambda_2}{\lambda} \right|^k, \quad k = 1, 2, \ldots . \tag{2.3}$$

Definition. *When $\hat{\alpha}_0 \neq 0$ define the factor s to be*

$$s = \frac{\lambda}{\hat{\alpha}_0^T \chi}. \tag{2.4}$$

Combining the asymptotic approximation above

$$\hat{\alpha}_0^T \hat{P}^{t-2} \mathbf{1}\ \approx \lambda^{t-2} \hat{\alpha}_0^T \chi = \lambda^{t-1} s^{-1}$$

with (2.2) gives the approximate formula

$$\begin{aligned} E(\theta) &\approx 1 + \hat{\alpha}_0^T \mathbf{1}\ + s^{-1}(\lambda^2 + \lambda^3 + \ldots) \\ &\approx \frac{1}{s} \frac{1}{1-\lambda} \end{aligned} \tag{2.5}$$

assuming $1 + \hat{\alpha}_0^T \mathbf{1}\ \approx s^{-1}(1+\lambda)$.

Of course only in special cases is it possible to contemplate calculating the factor s. One of these is when $\hat{P}$ has a simple and regular structure. Another is when N is relatively small, on the order of a few thousand. In this case, the Power Method for calculating eigenvalues and eigenvectors is effective, (Varga, 1963), see section 4.

2.2 *Estimating the factor s*

The parameters λ and s have the following interpretations. After several iterations, an arbitrary non-zero starting distribution $\hat{\alpha}_0$ tends to the invariant distribution ω, $\hat{\alpha}_0^T \hat{P}^t \to \omega^T$ as $t \to \infty$. But for a distribution that is approximately ω, the equation $\omega^T \hat{P} = \lambda \omega^T$ shows that λ fraction of the probability mass is retained in the non-goal states while $1 - \lambda$ escapes to the goal states. Thus λ is the asymptotic probability of remaining in the non-goal states on each iteration.

While λ gives the average rate of retention of the probability mass in the non-goal states, the normalized vector $\chi / \|\chi\|_1$ lists the retention rates per non-goal state. Indeed, for a primitive $\hat{P}$, $\hat{P}^t \to \lambda^t \chi \omega^T$ as $t \to \infty$ (see (Seneta, 1981)). But this limiting rank one matrix has the value $\lambda \chi^i$ as it's i^{th} row sum, which is the probability of retention in the non-goal states on the next iteration when the process is in the i^{th} non-goal state. Therefore $\hat{\alpha}_0^T \chi$ is proportional to the probability of retention in the non-goal states for the distribution $\hat{\alpha}_0$. Thus s is large when $\hat{\alpha}_0^T \chi$ is small and that occurs if the distribution α_0 leads to the goal with high probability. Conversely, s is small if starting is not likely to find a goal, rather the search process defined by $\hat{P}$ is the better bet.

The parameter s can be estimated as follows. Define the angle ϕ between ω and χ by $\cos\phi = \omega^T \chi / \|\omega\| \, \|\chi\|$ (Euclidean norm) and similarly let δ be the angle between $\hat{\alpha}_0$ and χ. Then combining $1 = \omega^T \chi$ and $\frac{s}{\lambda} = 1 / \hat{\alpha}_0^T \chi$ gives

$$\frac{s}{\lambda} = \frac{\|\omega\| \cos\phi}{\|\hat{\alpha}_0\| \cos\delta}. \tag{2.6}$$

This shows that $\frac{s}{\lambda}$ is the ratio between the projections of the probability vector ω and the vector $\hat{\alpha}_0$ onto χ.

Theorem 2.2.1. *If the row sums of $\hat{P}$ are equal, then $s \geq \lambda$. If in addition the initialization process α_0 has at least $1-\lambda$ probability of finding a goal state, then $s \geq 1$.*

Proof. Under the hypothesis λ is the common row sum and, properly normalized, $\chi = \mathbf{1}$ the vector of 1's. Since $\hat{\alpha}_0$ is a subprobability vector, $\hat{\alpha}_0^T \chi \leq 1$ so $s \geq \lambda$. If in addition $\hat{\alpha}_0^T \chi = \hat{\alpha}_0^T \mathbf{1} \ \leq \lambda$, then $s \geq 1$. ∎

Theorem 2.2.2. *If $\hat{\alpha}_0$ has equal components and $\phi < \delta$, then $s \geq \lambda$. If in addition the initialization process α_0 has at least $1-\lambda$ probability of finding a goal state, then $s \geq 1$.*

Proof. Since ω is a probability vector, it has minimum Euclidean norm when all its components are equal (to $\frac{1}{N-g}$); hence $\|\omega\| \geq \frac{1}{\sqrt{N-g}}$. Under the hypothesis, $\hat{\alpha}_0$ has the form $\sigma \mathbf{1}$ for some $\sigma \leq \frac{1}{N-g}$. Hence $\|\hat{\alpha}_0\| = \sigma\sqrt{N-g} \leq \|\omega\|$. This proves the first assertion. If in addition $\hat{\alpha}_0^T \mathbf{1} \ \leq \lambda$, then also $\sigma \leq \frac{\lambda}{N-g}$ so that $\|\hat{\alpha}_0\| \leq \frac{\lambda}{\sqrt{N-g}} \leq \lambda\|\omega\|$. Now the second assertion follows. ∎

Theorem 2.2.3. *If $\hat{P}$ is symmetric and $\hat{\alpha}_0$ has equal components then $s \geq \lambda$. If also $\hat{P}$ has unequal row sums then $s > \lambda$. If the initialization process α_0 has at least $1-\lambda$ probability of finding a goal state, then these inequalities may be replaced by $s \geq 1$ and $s > 1$ respectively.*

Proof. For $\hat{P}$ symmetric, χ is a scalar multiple of ω so that $\phi = 0$. Since $\hat{\alpha}_0$ has equal components, $\|\hat{\alpha}_0\| \leq \|\omega\|$ as above and hence $s \geq 1$. If also $\hat{P}$ has unequal row sums, then $\delta \neq 0$, and so $s > 1$. The additional assertions when $\hat{\alpha}_0^T \mathbf{1} \ \leq \lambda$ follow as in the proof above. ∎

Starting a Monte Carlo method uniformly at random gives rise to an initialization vector $\hat{\alpha}_0$ vector with the equal components $\frac{1}{N}$.

Obviously s can be made arbitrarily large by choosing the components of $\hat{\alpha}_0$ arbitrarily small. We show by example that s can be arbitrarily small (positive). Consider the matrix

$$\hat{P} = \begin{pmatrix} .5 & 2\epsilon^2 \\ .5 & .5 \end{pmatrix}.$$

Here $\lambda = 0.5 + \epsilon$, $\omega^T = \frac{1}{0.5+\epsilon}(0.5 \quad \epsilon)$, and $\chi^T = (0.5 + \epsilon)(1 \quad \frac{1}{2\epsilon})$. As $\epsilon \to 0$, $\lambda \to 0.5$, but the second component of χ grows unboundedly. It follows that s can be made arbitrarily small.

2.3. *Equally Likely Trials*

As a special case consider the optimization process in which on each iteration the next Markov state is selected uniformly at random. Then the transition probability matrix P has identical elements at every position.

$$P = \begin{bmatrix} p & p & \cdots & p \\ \vdots & \vdots & & \vdots \\ p & p & \cdots & p \end{bmatrix}$$

where $p = 1/N$ and N is the number of Markov states. If states 1 through g are the target states then $\hat{P}$ is the $N-g \times N-g$ matrix all of whose elements are p. In this case the $N-g$ vector $\mathbf{1}$ and its transpose are the positive right and left eigenvectors of $\hat{P}$ respectively, so

$$\omega = \frac{1}{N-g}\mathbf{1} \quad \text{and} \quad \chi = \mathbf{1} \ . \tag{2.7}$$

The principal eigenvalue λ is the common row sum

$$\lambda = (N-g)p = 1 - \frac{g}{N}. \tag{2.8}$$

Now suppose the process starts uniformly at random so that $\alpha_0^T = (\frac{1}{N}, \ldots, \frac{1}{N})$ and $\hat{\alpha}_0 = \frac{1}{N}\mathbf{1}$. Then $\hat{\alpha}_0^T \chi = \hat{\alpha}_0^T \mathbf{1} \ = \lambda$, and for $t \geq 3$,

$$\Pr(\theta \geq t) = \hat{\alpha}_0^T \hat{P}^{t-2} \mathbf{1} \ = \hat{\alpha}_0^T \lambda^{t-2} \mathbf{1} \ = \lambda^{t-1},$$

exactly. The expected hitting time for equally likely trials is

$$E(\theta) = 1 + \lambda + \sum_{k=3}^{\infty} \lambda^{k-1} = \frac{1}{1-\lambda}. \tag{2.9}$$

The s-factor for equally likely trials is

$$s = \frac{\lambda}{\hat{\alpha}_0^T \chi} = 1, \tag{2.10}$$

in agreement with Theorem 2.2.1.

2.4. *Non-Stationary Markov Chains*

As above we assume the Monte Carlo method corresponds to a Markov chain X_t, $t = 1, 2, \ldots$, but now we allow the transition probabilities p_t^{ij} to depend on the iteration index t. In this case the expected hitting time is given by an infinite series. As above assume the target states are indexed $1, \ldots, g$, and let E_i^t denote the expected incremental hitting time to one of $\mathcal{P}_1, \ldots \mathcal{P}_g$, starting from state $\mathcal{P}_i$ at time t, that is E_i^t is the expected increment in the hitting time beyond t. Conditioning on the possible transitions from state i, $i = g+1, \ldots, N$, at time t,

$$\begin{aligned} E_i^t &= p^{i1}(t) + \ldots + p^{ig}(t) + p^{ig+1}(t)(E_{g+1}^{t+1} + 1) + \cdots + p^{iN}(t)(E_N^{t+1} + 1) \\ &= p^{i1}(t) + \cdots + p^{iN}(t) + p^{ig+1}(t)E_{g+1}^{t+1} + \cdots + p^{iN}(t)E_N^{t+1} \\ &= \mathbf{1} \ + \hat{p}^{i\cdot}(t) \cdot \mathbf{E}^{t+1} \end{aligned}$$

where $\hat{p}^{i\cdot}(t)$ is the deleted i^{th} row vector of the time t transition matrix $P(t)$, $i = g+1, \ldots, N$, and $\mathbf{E}^{t+1}$ is the expectation vector. In matrix notation

$$\mathbf{E}^t = \mathbf{1} \ + \hat{P}(t)\mathbf{E}^{t+1}, \qquad t = 1, 2, \ldots,$$

where $\mathbf{1}$ is the $N - g$ vector of 1's and $\hat{P}(t)$ is the matrix gotten from $P(t)$ by deleting the latter's first g rows and columns. The solution $\mathbf{E} = \mathbf{E}^1$ is given by induction on the equation above,

$$\mathbf{E} = 1 + \sum_{t=1}^{\infty} \prod_{j=1}^{t} \hat{P}(j)\mathbf{1} \tag{2.11}$$

where $\prod_{j=1}^{t} \hat{P}(j) = \hat{P}(1)\hat{P}(2)\cdots\hat{P}(t)$. This series can diverge.

Just as in the stationary case, the hitting time distributions $\Pr(\theta \geq k)$ may be obtained numerically from (2.1) by modifying the transition matrix so that the target states are absorbing.

2.5. *Sampling Without Replacement— Perfect Recall*

For reference, we include the hitting time calculation for a method which is not equivalent to a Markov chain in that each new trial uses the complete history of previous trials. In particular each previously tried point is remembered and not tried again. We include this example because we will see that even with such complete information parallel Monte Carlo methods can nonetheless be substantially spedup (cf. Section 3.5).

Suppose the Monte Carlo method is arranged so that the probability of finding an optimal state on the t^{th} iteration is hyperbolically increasing, $\frac{p}{1-(t-1)p}$ for $p = 1/N$, $N = \text{card}(D)$, and $1 \leq t \leq N$. This would arise if visited states are systematically eliminated from further consideration. Then it is easy to see that

$$\Pr(\theta = t) = p, \qquad 1 \leq t \leq N. \tag{2.12}$$

In this case the expected hitting time is finite

$$\Pr(\theta \geq t) = \begin{cases} 0, & t > N, \\ kp, & 1 \leq t = N + 1 - k \leq N \\ Np, & t \leq 1, \end{cases} \tag{2.13}$$

and so from (1.3)

$$E(\theta) = \frac{N(N+1)}{2} p = \frac{N+1}{2}. \tag{2.14}$$

3 Parallel Monte Carlo Methods

We next examine the behavior of Monte Carlo optimization algorithms when multiple copies are run in parallel. We assume that m identical processes are run simultaneously and independently of each other except that any one process can flag them all to stop. The variable t now counts "wall clock time," i.e., the same individual time for each process, not their cumulative time.

This multiprocess can be viewed as an m-tuple $\mathcal{X}_t = (X_t^1, \ldots, X_t^m)$ of the m-individual processes defined on their m-fold product probability space. This holds for both the stationary and non-stationary chains. By the *multi-process hitting time* Θ we mean the random variable equal to the first time t that any one of the m-individual processes is in G,

$$X_t^k \in G, \quad \text{for some } 1 \leq k \leq m$$

and

$$X_\tau^i \notin G, \quad 1 \leq i \leq m, \quad \tau < t.$$

Or equivalently

$$\Theta = \min_{1 \leq i \leq m} \{\theta_i\},$$

where θ_i is the hitting time of the i^{th} process X_t^i, $i = 1, \ldots, m$. We use the notation E or E_1 for $E(\theta)$ and E_m for $E(\Theta)$ when the number of processes is m.

3.1. *Multiprocess Expectation and Hitting Time Distribution Calculation*

Since the multi-process is a Markov process, $\mathcal{X}_t = (X_t^1, \ldots, X_t^m)$ which is the Cartesian product of m identical, independent Markov processes X_t, we may take as a state of the multi-process, the Cartesian product of single-process states. The cardinality of the

product space is N^m and the multi-process transition matrix P is correspondingly large. Its elements are

$$\begin{aligned} p^{i_1,\ldots,i_m}_{j_1,\ldots,j_m}(t) &= \Pr(\mathcal{X}_{t+1} = (x_{j_1},\ldots,x_{j_m}) \mid \mathcal{X}_t = (x_{i_1},\ldots,x_{i_m})) \\ &= \Pr(X^1_{t+1} = x_{j_1} \mid X^1_t = x_{i_1}) \cdots \Pr(X^m_{t+1} = x_{j_m} \mid X^m_t = x_{i_m}) \\ &= p^{i_1 j_1}(t) \cdots p^{i_m j_m}(t). \end{aligned}$$

Now the multi-process expectations and hitting time distributions can be calculated as before using instead this Cartesian product transition matrix. Unfortunately the resulting matrix products quickly become unwieldy. In fact, except in special cases, P already is. However there is a much easier way to calculate the complementary hitting time distribution owing to the fact that the processes are identical and independent.

3.2. *The Multi-process Complementary Hitting Time Distribution*

It is evident that the event $\Theta \geq t$ is equivalent to the event

$$\theta_1 \geq t \quad \text{and} \quad \theta_2 \geq t \quad \text{and} \cdots \text{and} \quad \theta_m \geq t.$$

Therefore by independence we have the following.

Proposition. For m identical, independent processes

$$\Pr(\Theta \geq t) = (\Pr(\theta \geq t))^m, \quad t = 1, 2, \ldots \tag{3.1}$$

and so

$$E_m = E(\Theta) = \sum_{t=1}^{\infty} (\Pr(\theta \geq t))^m. \tag{3.2}$$

We next calculate the expectation of Θ for the hitting time probability distributions of the previous section. From these calculations we will see that the parallel speedup prospects of Monte Carlo algorithms are excellent. By speedup we mean

$$\text{Speed-Up} = \frac{E(\theta)}{E(\Theta)}. \tag{3.3}$$

Main Theorem. *Let $\mathcal{X}_t = (X_t^1, \ldots, X_t^m)$ be an m-fold multi-process of m identical independent stationary Markov chains X_t^i, $t = 1, 2, \ldots$, $i = 1, \ldots, m$. Let λ be the principal eigenvalue of the single process deleted transition probability matrix $\hat{P}$ with $\hat{P}$ primitive. If the starting state is not in the goal with certainty, then*

$$\begin{aligned} \textit{Speedup} &= s^{m-1}\frac{1-\lambda^m}{1-\lambda} + O(1-\lambda^m) \\ &\approx m s^{m-1} \qquad \text{as } \lambda \approx 1. \end{aligned} \tag{3.4}$$

Proof. By eq.(2.1) and the Theorem of section 2.1 we known that for $k \geq 3$

$$(\lambda s^{-1} - h(k-2)\delta^{k-2})\lambda^{k-2} \leq \Pr(\theta \geq k) \leq (\lambda s^{-1} + h(k-2)\delta^{k-2})\lambda^{k-2}$$

for $h(\cdot)$ a fixed polynomial and $\delta = \frac{|\lambda_2|}{\lambda} < 1$. Therefore

$$(\lambda s^{-1} - h(k-2)\delta^{k-2})^m(\lambda^m)^{k-2} \leq \Pr(\Theta \geq k) \leq (\lambda s^{-1} + h(k-2)\delta^{k-2})^m(\lambda^m)^{k-2}.$$

By the Root Test the series $\sum h(k-2)\delta^{k-2}\lambda^{k-2}$ converges for all $0 \leq \lambda \leq 1$, say to $S(\lambda)$. Further as $\lambda \to 1$, $S(\lambda) \to S(1)$ and $S(1)$ is finite. Therefore from (2.2)

$$1 + \hat{\alpha}_0^T \mathbf{1} \ + \frac{s^{-1}\lambda^2}{1-\lambda} - S(\lambda) \leq E(\theta) \leq 1 + \hat{\alpha}_0^T \mathbf{1} \ + \frac{s^{-1}\lambda^2}{1-\lambda} + S(\lambda), \qquad 0 \leq \lambda < 1. \tag{3.5}$$

Also the series

$$\begin{aligned} &\sum_{k=3}^{\infty}(\lambda s^{-1} \pm h(k-2)\delta^{k-2})^m(\lambda^m)^{k-2} \\ &\qquad = \sum_{k=3}^{\infty}\sum_{j=0}^{m}\binom{m}{j}\lambda^{m-j}s^{j-m}(\pm h(k-2))^j(\delta^j)^{k-2}(\lambda^m)^{k-2} \\ &\qquad = \sum_{j=0}^{m}\binom{m}{j}\lambda^{m-j}s^{j-m}\sum_{k=3}^{\infty}(\pm h(k-2))^j(\delta^j)^{k-2}(\lambda^m)^{k-2} \end{aligned}$$

converges because for each $j = 0, \ldots, m$ the series

$$S_{\pm}^{(j)}(\lambda) = \sum_{k=3}^{\infty}(\pm h(k-2))^j(\delta^j)^{k-2}(\lambda^m)^{k-2}$$

does. For $j \neq 0$, $S_{\pm}^{(j)}(\lambda)$ is finite for all $0 \leq \lambda \leq 1$, but for $j = 0$, since $s^{-1} \neq 0$,

$$\lambda^m s^{-m} S^{(0)}(\lambda) = \lambda^m s^{-m} \sum_{k=3}^{\infty} (\lambda^m)^{k-2} = \frac{s^{-m}\lambda^{2m}}{1-\lambda^m} \to \infty \text{ as } \lambda \to 1.$$

Letting $b_\lambda^{\pm}$ denote the finite sum

$$b_\lambda^{\pm} = \sum_{j=1}^{m} \lambda^{m-j} s^{j-m} \binom{m}{j} S_{\pm}^{(j)}(\lambda),$$

and combining inequalities above we get

$$\frac{1 + \hat{\alpha}_0^T \mathbf{1} \ + \frac{s^{-1}\lambda^2}{1-\lambda} - S(\lambda)}{1 + (\hat{\alpha}_0^T \mathbf{1} \ )^m + \frac{s^{-m}\lambda^{2m}}{1-\lambda^m} + b_\lambda^+} \leq \frac{E(\theta)}{E(\Theta)} \leq \frac{1 + \hat{\alpha}_0^T \mathbf{1} \ + \frac{s^{-1}\lambda^2}{1-\lambda} + S(\lambda)}{1 + (\hat{\alpha}_0^T \mathbf{1} \ )^m + \frac{s^{-m}\lambda^{2m}}{1-\lambda^m} - b_\lambda^-}. \tag{3.6}$$

Put $B = b_\lambda^- + s^{-m}(1+\lambda^m) - 1 - (\hat{\alpha}_0^T \mathbf{1} \ )^m$. Since $s^{-m}(1+\lambda^m) + \frac{s^{-m}\lambda^{2m}}{1-\lambda^m} = \frac{s^{-m}}{1-\lambda^m}$, the third member of this chain of inequalities can be rewritten as

$$\frac{(s^{-1}(1+\lambda) + \frac{s^{-1}\lambda^2}{1-\lambda})(1-\lambda^m)}{s^{-m} - B(1-\lambda^m)} + \frac{(1 + \hat{\alpha}_0^T \mathbf{1} \ - s^{-1}(1+\lambda) + S(\lambda))(1-\lambda^m)}{s^{-m} - B(1-\lambda^m)}.$$

Evidently the second term tends to 0 with $1 - \lambda^m$. Expand the first term as a geometric series in powers of $(1-\lambda^m)$ to get

$$\frac{s^{-1}(1-\lambda^m)}{s^{-m}(1-\lambda)}(1 + s^m\lambda^{-m}B(1-\lambda^m) + (s^m\lambda^{-m}B(1-\lambda^m))^2 + \ldots)$$

showing that the speedup is bounded above as claimed. Evidently the first member of (3.6) behaves in like manner and so the estimate (3.4) is proved. ∎

Remark. It should be kept in mind that for any given problem, λ is fixed (usually very nearly equal to 1 in our experience) and hence this is not an asymptotic result in λ. Similarly s is fixed for a given problem (usually slightly greater than 1 in our experience) and while s is not a function of λ, by modifying a problem in such a way that λ increases, it may happen that also s decreases.

Definition. On the basis of this result, we define the speedup of a Monte Carlo method to be *accelerated* when $s > 1$.

Figure 2 depicts the function $s^{m-1}(\frac{1-\lambda^m}{1-\lambda})$ as a function of m for fixed $\lambda = .99$, and 3 different values of s, $s = 1.02$, 1.0, and 0.98. As a function of m

$$\frac{1-\lambda^m}{1-\lambda} \longrightarrow \frac{1}{1-\lambda}, \qquad m \to \infty$$

so that its contribution to speedup chokes off for large m. By contrast the term due to s is exponentially increasing with m when $s > 1$ so that the actual speedup can be *superlinear*.

Remark. Of course E_m can never be less than 1 so that actual speedup can never exceed E_1; in other words, the term $O(1-\lambda^m)$ does not go away with increasing m.

Speedup vs Number of Processors for $\lambda = 0.99$

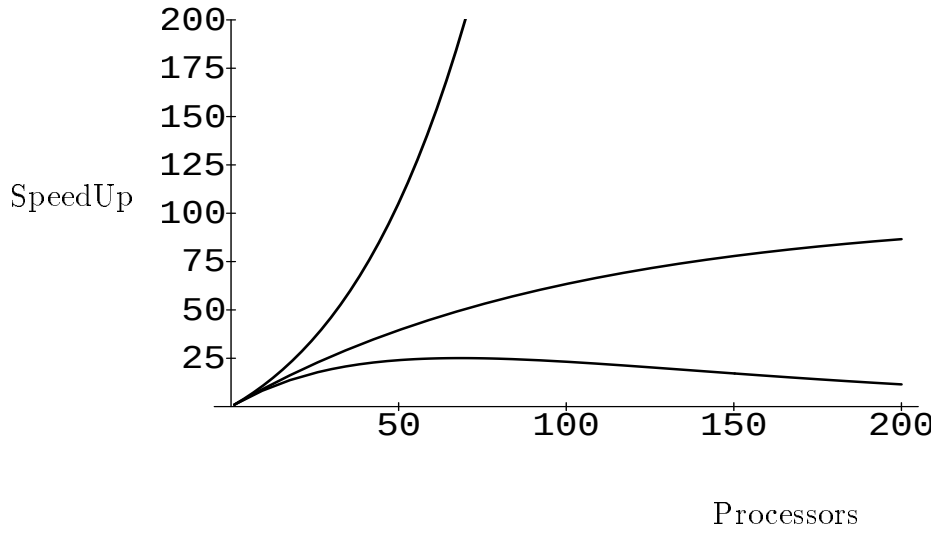

fig. 2

3.3. *Equally Likely Trials Revisited*

From section 2.3 for equally likely trials with uniform random starting $s = 1$, $\hat{\alpha}_0^T \mathbf{1} = \lambda$, and $\Pr(\theta \geq t) = \lambda^{t-1}$. Hence (3.6) holds with $S(\lambda) = b_\lambda^{\pm} = 0$. The multi-process exact hitting time is $E(\Theta) = 1/(1-\lambda^m)$ and the speedup is

$$\frac{1-\lambda^m}{1-\lambda} \longrightarrow \begin{cases} m, & \lambda \to 1, \\ \frac{1}{1-\lambda}, & m \to \infty. \end{cases}$$

This is illustrated by the middle curve in fig. 2.

3.4. *Sampling Without Replacement*

Here the single process hitting time distribution is $\Pr(\theta = t) = p$, $1 \leq t \leq N$, according to equation (2.12) and the complementary hitting time distribution is given by equation (2.13). Therefore the multi-process hitting time distribution is

$$\Pr(\Theta \geq t) = \begin{cases} 0, & t \geq N, \\ (kp)^m, & 1 \leq t = N + 1 - k \leq N, \\ (Np)^m, & t \leq 1. \end{cases}$$

where $N = 1/p$. The expected multi-process hitting time is

$$E(\Theta) = (Np)^m + (N-1)^m p^m + \cdots + 2^m p^m + p^m = p^m \sum_{k=1}^{N} k^m.$$

But the sum in the latter member is known to be a polynomial $r(N)$ in N of degree $m+1$ and leading coefficient equal to $1/(m+1)$. Therefore the speedup is

$$\frac{\frac{1}{2}(N+1)}{\frac{1}{m+1}p^m(N^{m+1} + q(N))} = \frac{m+1}{2} \frac{1 + \frac{1}{N}}{1 + \frac{1}{N^{m+1}} q(N)}$$

where $q(N)$ is a polynomial of degree m. Therefore as $N \to \infty$,

$$\text{Speedup} \to \frac{m+1}{2}.$$

The speedup is only about half linear here because by eliminating previously unsuccessful tries, the search algorithm is utilizing more and more information as it proceeds. Yet the independent multiple processes gain no advantage from their mutual information during a run since they do not communicate and hence do not share it.

4 Computational Results on Some Specific Problems

We give here some results of the foregoing developments applied to three problems. The first two are chosen sufficiently small in size that s and λ may be exactly determined and hence serve to illustrate the extent of agreement between empirical results and theory. Again for illustrative purposes, one is a single state Markov chain and the other is a multiple state chain – a Genetic Algorithm, both are homogeneous Markov chains. The third problem is the inverse problem of fractal geometry, whose analytical solution has so far eluded researchers. (Diaconis and Shahshahani, 1986), (Vrscay and Roehrig, 1989). We give here only the speedup results, the details of the solution method appear elsewhere, (Shonkwiler, Mendivil, and Deliu, 1991).

In the last subsection of this section we compute the exact speedup for a simple non-homogeneous method and show by example that infinite speedup is possible in this case.

4.1. *Password Problem*

Assume that a J character password chosen from an alphabet of M symbols is to be found. Trying a proposed solution results in either failure or success, there are no hints. The domain D consists of all strings of J legal symbols, $\text{card}(D) = M^J$, and for $x \in D$ the objective function will be taken as

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is correct} \\ 0, & \text{if } x \text{ is incorrect.} \end{cases}$$

In reality, except for the extreme nature of its objective function, the password problem is typical of very many problems encountered in practice. Indeed, any problem $v = f(x_1, \ldots, x_n)$ defined on a rectangle

$$a_i \le x_i \le b_i, \qquad i = 1, \ldots, n$$

in n-dimensional Euclidean space $\mathbf{R}^n$ has this form computationally. For if the binary floating point representations of each component consisting of md mantissa digits, ed exponent digits, and one sign digit

$$x_i = s^i b_1^i b_2^i \dots b_{md}^i e_1^i \dots e_{ed}^i$$

are concatenated, there results a password problem with $J = n(md + ed + 1)$ characters from the alphabet $\{0, 1\}$ of size $M = 2$. In this way, such a problem with a general objective function becomes a *password problem with hints* (to the extent that the broadness of the basins of the objective help locate the global minimum).

Our scheme for generating new trial solutions from a given one will be *one character uniform replacement*, that is if x is the present solution attempt, select a character position $1, 2, \dots, J$ at random and replace the letter of x at that position by a randomly chosen letter from the alphabet; this is changing one component at a time looking for improvement. The "neighborhood" of each point x in this scheme consists of MJ points. By contrast, in equally likely trials all M^J points of the domain are in the neighborhood of every x.

The transition probability matrix for one character uniform replacement has for each row x, unless x is the solution, a zero in every column corresponding to a $y \in D$ differing from x in two or more positions. The probability for those $y \in D$ differing in exactly one position from x is $1/(JM)$, and the probability that x itself is reselected is $1/M$. This transition probability matrix, P, is symmetric, the deleted transition probability matrix $\hat{P}$, is also symmetric and has unequal row sums. The latter follows since some states of $\hat{P}$ lead to the goal while others do not. Therefore Theorem 2.2.3 applies for a uniformly selected starting state and we can get accelerated speedup for this problem.

With the choices $J = 4$, and $M = 5$ the matrix P is 625×625 and $\hat{P}$ is 624×624. It is possible to calculate all the relevant optimization characteristics exactly. Assuming a uniformly selected starting state, in Table I we show the principle eigenvalue λ, the s-factor s, the exact expected hitting time E, and the exact expected hitting times E_2, E_4, E_8 for

4.2 *Genetic Algorithm Solver for the Sandia Mountain Problem*

Let the domain D be the set of integers $D = \{0, 1, ..., N\}$, card$(D) = N + 1$, and let the objective function be

$$f(x) = \begin{cases} \frac{N-x}{N-1}, & x = 1, 2, \ldots, N \\ -1, & x = 0, \end{cases}$$

i.e. a long gradual uphill slope from $x = N$ to $x = 1$, but then a steep drop at $x = 0$, see fig. 4.

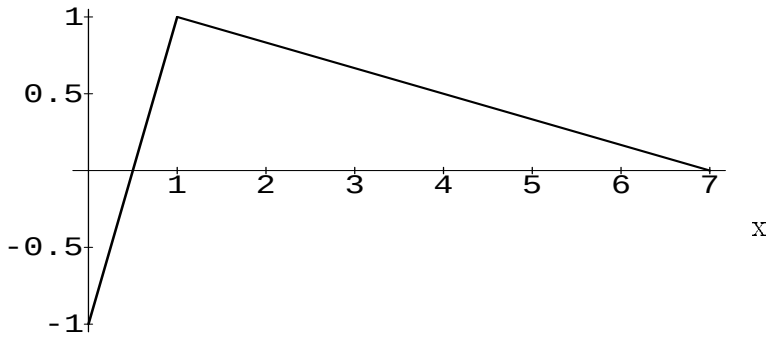

fig. 4

The global minimum of -1 occurs at $x = 0$ and this minimum has a basin of two domain points. There is also a local minimum of 0 occurring at $x = N$. This basin is of size N. To keep the example within reasonable size let $N = 7$ and represent the $N + 1 = 8$ domain values in binary

$$0 \leftrightarrow (000)_2, \quad 1 \leftrightarrow (001)_2, \ldots, 7 \leftrightarrow (111)_2.$$

We employ a standard Genetic Algorithm (Goldberg, 1989)with a population size of 2, reproductive success taken in proportion to fitness ϕ which will be defined as

$$\phi(x) = 2 - f(x), \quad x \in D,$$

crossover based on bit strings and a bit mutation rate of $p_m = 0.001$. The number of distinct populations is $\frac{8 \cdot 9}{2} = 36$ and is ordered as $< 0,0 >, \ldots, < 0,7 >, < 1,1 >, \ldots, < 7,7 >$.

An iteration of the algorithm will consist of: (1) a reproduction of the present population, each "individual" in proportion to its fitness; let P_R be the 36×36 transition probability matrix for this process. Next (2) a crossover or mating process based on bit strings. Note that the mate selection matrix is just the identity because the population size is 2. For this 3 bit example the two crossover sites, between bits 1 and 2 or between bits 2 and 3, are chosen equally likely. Let P_c denote the 36×36 matrix for this process. Then (3) a mutation in which one of the two population members is chosen equally likely and each bit of the chosen member is reversed ($0 \to 1$ and $1 \to 0$) with probability p_m independently; P_m denotes the resulting 36×36 transition matrix. Finally (4) the required function evaluations are performed to obtain the next generation's fitness and to update the "best" random variable B_t.

These processes may be elaborated as follows. During the reproduction process the population $< i,j >$ will become one of the populations $< i,i >$ or $< i,j >$, or $< j,j >$. If $\phi_i \equiv \phi(i)$ is the fitness of $i \in D$, then the probability of obtaining $< i,i >$ is $\left(\frac{\phi_i}{\phi_i + \phi_j}\right)^2$, of $< i,j >$ is $2\left(\frac{\phi_i \phi_j}{\phi_i + \phi_j}\right)$ and of $< j,j >$ is $\left(\frac{\phi_j}{\phi_i + \phi_j}\right)^2$. During crossover the population $< i,j >$ with corresponding bit strings $i = b_1 b_2 b_3$ and $j = B_1 B_2 B_3$ will become

$$b_1 B_2 B_3 \quad \text{and} \quad B_1 b_2 b_3 \quad \text{with probability } \frac{1}{2}$$

or

$$b_1 b_2 B_3 \quad \text{and} \quad B_1 B_2 b_3 \quad \text{with probability } \frac{1}{2}.$$

Finally, under mutation, the population $< i = (b_1 b_2 b_3)_2,\ j = (B_1 B_2 B_3)_2 >$ will become, with prime denoting bit complementation,

$$b_1 b_2 b_3 \quad \text{and} \quad B_1 B_2 B_3 \quad \text{with probability } (1 - p_m)^3$$

or

$$b_1 b_2 b_3' \quad \text{and} \quad B_1 B_2 B_3 \quad \text{with probability } \frac{1}{2}(1-p_m)^2 p_m$$

and so on until

$$b_1 b_2 b_3 \quad \text{and} \quad B_1' B_2' B_3' \quad \text{with probability } \frac{1}{2} p_m^3.$$

The 36×36 overall transition probability matrix P is the product of its three component parts reproduction, crossover and mutation by the independence of these processes, and works out to be

$$P = P_R P_c P_m = \begin{bmatrix} .729 & .081 & \dots & .000 \\ .425 & .324 & \dots & .000 \\ \vdots & \vdots & & \vdots \\ .000 & .000 & \dots & .729 \end{bmatrix}.$$

Note that each of the 8 populations $< 0,0 >, \dots, < 0,7 >$ containing 0 solve the problem. Therefore the deleted transition matrix $\hat{P}$ is 28×28 and omits the first 8 rows and columns of P.

As in the first example we may calculate the optimization characteristics from the transition probability matrix. These data are shown in Table II and fig. 5. One sees that a real speedup of over 11 is achieved using 8 processors.

Remark. This example affords a simple explanation as to why superlinear speedup is possible. There is a certain (relatively large) probability p that the process will be in the sub-optimal state $x = 7$. (Here $p = .185$ and is the last component of the normalized left eigenvector ω for $\hat{P}$.) By contrast the probability q that the process will be in state $x = 1$, the threshold of the basin for the solution, is small ($q = .029$, the first component of ω). However for m independent processes, the probability they *all* will be in state $x = 7$ is p^m – small for large m, while the probability that*at least one* of the m processes is in state $x = 1$ increases with m, $1 - (1-q)^m$. But it only takes one process to find the solution.

The transition probability matrix is given by

$$p_{ij} = \begin{cases} g_{ij}a_{ij} & \text{if } i \neq j \\ 1 - \sum_{k \neq i} p_{ik} & \text{if } i = j \end{cases}$$

thus

$$P = \begin{bmatrix} 1 - \frac{1}{2}e^{-2\Delta/T} & \frac{1}{2}e^{-2\Delta/T} & 0 \\ 1/2 & 0 & 1/2 \\ 0 & \frac{1}{2}e^{-\Delta/T} & 1 - \frac{1}{2}e^{-\Delta/T} \end{bmatrix}.$$

From annealing theory (*loc. cit.*) the temperature T should vary with iteration count t according to the equation

$$T = \frac{C}{\ell n(t+1)}$$

where C is the depth of the deepest local non-global minimum. Here $C = 1$. Eliminating T gives the transition probabilities directly in terms of t, thus

$$p_{21}(t) = \frac{1}{2}e^{-\Delta \ell n(t+1)} = \frac{1/2}{(t+1)^{\Delta}} = \frac{1/2}{t+1}, \qquad t = 1, 2, \ldots,$$

and

$$p_{22}(t) = 1 - p_{21}(t) = 1 - \frac{1/2}{t+1}, \qquad t = 1, 2, \ldots.$$

To analyze this process we use equation (2.11) to calculate the expected hitting time. In general the iterates

$$\prod_{j=1}^{t} \hat{P}^j$$

quickly become intractable. Indeed the various terms of this product contain all the possible ways leading to state $x = 0$ in t iterations starting from a given state. Here however we estimate these probabilities. Hitting the goal at time k includes the possibility of remaining for $t = 1, 2, \ldots, k-2$ in state $x = 2$, then moving in two consecutive iterations to states $x = 1$ and $x = 0$. Therefore, the probability of hitting at time k is at least as large as

$$\begin{aligned} h_k &= \big(1 - \frac{1/2}{2}\big)\big(1 - \frac{1/2}{3}\big) \cdots \big(1 - \frac{1/2}{k-1}\big)\frac{1}{2}\frac{1}{k}\frac{1}{2} \\ &> \big(1 - \frac{1}{2}\big)\big(1 - \frac{1}{3}\big) \cdots \big(1 - \frac{1}{k-1}\big)\frac{1}{k}\frac{1}{4} \\ &= \frac{1}{4k(k-1)}, \quad k = 2, 3, \ldots. \end{aligned}$$

It follows that the expected hitting time from state 2 is at least as large as

$$\sum_{k=2}^{\infty} kh_k = \frac{1}{4}\sum_{k=2}^{\infty} \frac{1}{k-1} = \infty.$$

4.4. Inverse Fractal Problem

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ be a finite set of affine contraction maps of the unit interval $I = [0,1]$ into itself, that is maps of the form

$$w(x) = sx + a, \qquad 0 \leq x \leq 1.$$

Here the parameter $s < 1$ is the scale factor and the parameter a is the translation.

Associated with every such collection $\mathcal{W}$ is its *attractor* $A = A(\mathcal{W})$, a unique subset $A \subset I$ characterized by the selfcovering property,

$$A = \bigcup_{i=1}^{n} w_i(A).$$

Given $\mathcal{W}$, it is an easy matter to computational produce A as follows. Starting from the fixed point x^*, say of map w_1, choose $i_1 \in \{1, 2, \ldots, n\}$ at random and plot $x_1 = w_{i_1}(x^*)$. (To make the image easier to see, plot a short vertical line at x_1.) Now repeat this step with x_1 in place of x^* and $x_2 = w_{i_2}(x_1)$ in place of x_1, then repeat with x_2 in place of x_1, e.t.c. until say 10,000 points have been plotted. This construction is known as the *Random Iteration Algorithm* for constructing the attractor.

The *inverse or encoding problem* consists in finding an IFS $\mathcal{W}$ whose attractor A is given. One may convert it to an optimization problem by employing the notion of distance $h(A, B)$, between two attractors A, and B, (Shonkwiler, 1989). Then Monte Carlo search works on function systems $\mathcal{W}$ finding any whose associated attractor has zero distance from the desired one.

A multi-state, homogeneous, Monte Carlo search (a Genetic Algorithm) gave the speedup results shown in Table III and fig. 6. The goal was taken to be achieving a distance of 500 or less. For complete details, see (Shonkwiler, Mendivil, and Deliu, 1991).

Table III

Inverse Fractal Problem Speedups	
$m = 1$	$SU = 1$
$m = 2$	$SU = 2.59$
$m = 4$	$SU = 9.75$
$m = 8$	$SU = 238$
$m = 12$	$SU = 619$
$m = 24$	$SU = 709$

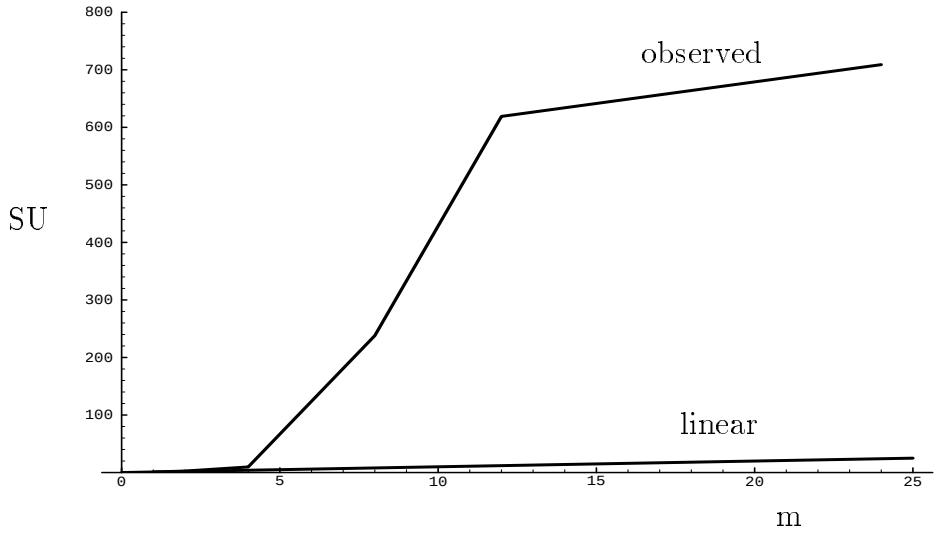

fig. 6

4.5. *Example*

Finally, we show by example that an annealer with infinite expectation can be converted to finite expectation when run in parallel. Consider the Sandia Mountain problem with $N = 1$ and the transition probability from state $x = 1$ to the goal $x = 0$ given by

$$p_{10}(t) = \frac{1}{t+1}, \quad p_{11}(t) = 1 - p_{10}(t), \qquad t = 1, 2, \ldots .$$

Then the event that the hitting time will be k occurs if and only if the process remains in state 1 for the first $k-1$ trials and moves to state 0 on the k^{th}; this has probability

$$(1-\frac{1}{2})(1-\frac{1}{3})\cdots(1-\frac{1}{k})\frac{1}{k+1} = \frac{1}{k(k+1)}, \qquad k = 2,3,\ldots .$$

and for $k = 1$, $\Pr(\theta = 1) = 1/2$. Therefore

$$\Pr(\theta \geq t) = \sum_{k=t}^{\infty} \frac{1}{k(k+1)} = \frac{1}{t}, \qquad t = 1,2,\ldots .$$

Then the single process expectation is infinite while the m multi-process expectation is

$$E_m = \sum_{t=1}^{\infty} \frac{1}{t^m} < \infty, \qquad m = 2,3,\ldots .$$

5 Conclusions

We have shown that superlinear speedup is possible with these types of algorithms. A given Monte Carlo method is characterized by its deleted transition probability matrix $\hat{P}$ and in particular its hitting time expectation depends on the complementary hitting time distribution. Two parameters, the principle eigenvalue λ of $\hat{P}$, and the s-factor, completely describe the tail of the hitting time distribution; asymptotically

$$\Pr(\theta \geq k) \approx s^{-1}\lambda^{k-1}.$$

The complementary hitting time distribution can be used to rigorously compare two Monte Carlo methods.

Hardware and software considerations for implementing multi-processed Monte Carlo algorithms are its strong point. Monte Carlo algorithms are typically easy to code and their parallelization couldn't be simpler. As for hardware, virtually any parallel architecture will do – massively parallel, hyper-cube, shared memory, distributed memory, etc.. The overhead taken by the IIP parallelization method is trivial in all but distributed memory

systems, and, even there, the overhead is insignificant for any problem whose parallel solution will require several minutes.

Finally one intriguing consequence of a superlinear parallel algorithm is the possibility of a new single process algorithm. In this case it is the running of multiple processes on a single processor machine. As we have shown, this technique will yield a faster converging uni-processor algorithm if the IIP method can be implemented with little additional overhead. But indeed this is possible by a technique we call *in code parallelism*. One sets up m data structures instead of one. Then the iteration loop is nested inside a loop that cycles through these data structures. The execution overhead is trivial, only memory limits the number of separate processes that can be handled.

References

Bertsekas, D., Tsitsiklis, J. (1989), "Parallel and Distributed Computation," Prentice Hall, Englewood Cliffs.

Chow, Y. S., Robbins, H., Siegmund, D. (1971), "The Theory of Optimal Stopping," Dover, New York.

Chung, K. (1967), "Markov Chains with Stationary Transition Probabilities," Springer, Berlin.

DARPA (1987), Proceedings of the DARPA Applied and Computational Mathematics Program Annual Meeting, Washington, D. C..

Diaconis, P. M. and Shahshahani, M. (1986), Products of Random Matrices and Computer Image Generation, *Contempory Math.*, **50**, 173–182.

Dorea, C. C. Y. (1990), Stopping Rules for a Random Optimization Method, *SIAM J. Control and Opt.,* **28**, no. 4, 841-850.

Geman, S. and Geman, D. (1984), Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images, *IEEE Trans*, **PAMI-6**, no. 6, 721-741.

Gidas, B. (1985), Nonstationary Markov chains and convergence of the annealing algorithm, *J. Stat. Phy.*, **39**, 73-131.

Goldberg, D. (1989), "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, Reading.

Hajek, B. (1988), Cooling schedules for optimal annealing, *Math. of Operations Research*, **13**, no. 2, 311-329.

Holland, J. (1975), "Adaptation in Natural and Artificial Systems", Univ. of Michigan Press, Ann Arbor.

Kemeny, J., Snell, L. (1960), "Finite Markov Chains", van Nostrand Co., Princeton.

Kirkpatrick, S., Gelatt, C., Vecchi, M. (1983), Optimization by simulated annealing, *Science* **220**, 671-680.

van Laarhoven, P., Aarts, E. (1987), "Simulated Annealing: Theory and Applications", D. Reidel, Boston.

Seneta, E. (1981), "Non-negative Matrices and Markov Chains", Springer-Verlag, New York.

Shonkwiler, R. (1989), An Image Algorithm for Computing the Hausdorff Distance Efficiently in Linear Time, *Inf. Proc. Letters* **30**, 87-89.

Shonkwiler, R., Mendivil, F. and Deliu, A. (1991), Genetic Algorithms for the 1-D Fractal Inverse Problem, "Proceedings of the Fourth International Conference on Genetic Algorithms," edited by Belew, R. and Booker, L., Morgan Kaufmann, San Mateo, CA, 495-501.

Varga, R. (1963), "Matrix Iterative Analysis", Prentice-Hall, Englewood Cliffs.

Vrscay, E. R. and Roehrig, C. J. (1989), Iterated function systems and the inverse problem of fractal construction using moments, "Computers and Mathematics," edited by Kaltofen, E. and Watt, S. M., Springer Verlag, Berlin, 250-259.

Zuckerman, D. (1986), Optimal Stopping in a Continuous Search Model, *J. Appl. Prob.*, **23**, 514-518.